AFRL-IF-RS-TR-2005-324
Final Technical Report
September 2005

CREATION AND MODELING OF ADAPTIVE AGENT SYSTEMS

Massachusetts Institute of Technology

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. K548

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2005-324 has been reviewed and is approved for publication

APPROVED: /s/

JOHN SPINA
Project Engineer

FOR THE DIRECTOR: /s/

JOSEPH CAMERA, Chief
Information & Intelligence Exploitation Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	SEPTEMBER 2005	Final Sep 00 – Oct 04

4. TITLE AND SUBTITLE
CREATION AND MODELING OF ADAPTIVE AGENT SYSTEMS

5. FUNDING NUMBERS
C - F30602-00-C-0216
PE - 62301E
PR - TASK
TA - 00
WU - 11

6. AUTHOR(S)
Oliver G. Selfridge and
Wallace Feurzeig

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Massachusetts Institute of Technology
545 Technology Square, NE 43-824
Cambridge Massachusetts 02139

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington Virginia 22203-1714

AFRL/IFED
525 Brooks Road
Rome New York 13441-4505

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2005-324

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: John Spina/IFED/(315) 330-4032/ John.Spina@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***

This project was dedicated to the development of multi-agent systems with nontrivial capabilities for adaptation and control in complex operational environments. The agents in such systems are based on control structures and processes that are inherently purpose-driven. The basic building blocks underlying the operation of purpose-driven agent systems are Elementary Adaptive Modules. EAMs have six elements: a purpose, an action, an action target, an activation, an evaluation function, and a set of control variables. EAMs can be passive or active. Passive EAMs, like servomechanisms, act in response to stimuli that are triggered in their external environment. Active EAMs, like hill-climbers, initiate probes in the outside world, evaluate the response, and change their behavior accordingly. EAM-based agents are cascaded to form multi-level systems whose control structures generate goal-oriented behaviors with greatly enhanced capabilities for adaptation and learning.

14. SUBJECT TERMS
Multi-Agent System, Purpose-Driven Agent Systems, Elementary Adaptive Modules

15. NUMBER OF PAGES
24

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

List of Figures

Acknowledgments

BBN software developer Brett Benyo implemented the EAM agent software used in the applications and made major contributions to their design. BBN senior scientist David Montana designed and applied the genetic programming methodology to the offline and online UAV learning developments. MIT graduate student Jacky Mallett designed and conducted several of the experiments on EAM-based intelligent traffic light agents.

Executive Summary

This project was dedicated to the development of multi-agent systems with nontrivial capabilities for adaptation and control in complex operational environments. The agents in such systems are based on control structures and processes that are inherently purpose-driven. The basic building blocks underlying the operation of purpose-driven agent systems are Elementary Adaptive Modules. EAMs have six elements: a purpose, an action, an action target, an activation, an evaluation function, and a set of control variables. EAMs can be passive or active. Passive EAMs, like servomechanisms, act in response to stimuli that are triggered in their external environment. Active EAMs, like hill-climbers, initiate probes in the outside world, evaluate the response, and change their behavior accordingly. EAM-based agents are cascaded to form multi-level systems whose control structures generate goal-oriented behaviors with greatly enhanced capabilities for adaptation and learning.

During the initial phase of the project, we constructed a basic set of elementary adaptive modules, including hill-climbers and servomechanisms, both singly and in combination. We demonstrated their capabilities for adaptive control in two trial applications calling for real-time adaptation: pole balancing, a manual skill task, and tracking a signal through noise, a complex signal filtering task. During the middle phase of the project, we implemented the first EAM-based multi-agent system, a versatile system for traffic light control in a complex and rapidly changing environment. The simulation showed that the behavior of complex traffic flows can be effectively controlled by continuously adapting algorithms to achieve purposes such as minimizing the average vehicle delay time.

During the last phase of the project, we developed an EAM-based multi-agent system for Unmanned Aerial Vehicle (UAV) surveillance in complex, dynamic, and uncertain operational environments. The UAV agents were multi-level hierarchical EAM structures. UAV sensors sought to detect targets and provide bearing and signal strength. UAV agents continually evaluated their performance in relation to their purposes and took appropriate actions. The UAV surveillance simulation included multiple clusters of targets, some with unknown locations. The simulation environment included objects that the UAVs were not programmed to handle in advance, such as winds that could push UAVs off course and hostile targets such as SAMs that could down the UAVs. Higher-level agents coordinated the collaborative operation of groups of individual UAV agents. The UAV agents demonstrated sophisticated target surveillance capabilities. They adapted their behavior and responded effectively in dynamically changing situations.

During the final year of the project, we integrated the use of genetic programming (GP) methods in the run-time simulation to support both off-line and on-line learning during the surveillance simulation. The GP learning approach demonstrated a capability for

generating new and improved control behaviors, resulting in further enhancement of UAV adaptive control performance. The software and documentation developed in the project is in our project Website, http://omar.bbn.com/MIT/AdaptiveAgents/.

Introduction

The goal of this project was to develop and apply a software technology for creating and modeling agent systems with strong capabilities for adaptation and learning, so as to enable agents to address new circumstances and changing needs while performing complex tasks. The idea behind such agents is that *they take responsibility* to try and satisfy some purpose. Agents, in this conception, are inherently purpose-driven entities. Their actions are guided by their purposes rather than by fixed condition-action rules. They must operate in uncertain environments and respond to unanticipated contingencies; thus they need to be inherently adaptive. They are essentially different from current agent systems, which incorporate pre-defined responses to pre-determined external stimuli.

We developed elementary adaptive controls as the basic constituents of agent structures in software systems for accomplishing real-life tasks. In our model, agents are quasi-autonomous objects—they are entities dealing with other agents; but within each agent there are sub-agents that undertake the control of lower sub-agents, all the way down to the control of external actions. In this view, agents are quasi-autonomous entities capable of dealing with other agents as well as the outside world. The basic elements of these agent structures are adaptive controls with explicitly embedded purposes. In our design, agents are constructs whose constituents are *elementary adaptive modules* (EAMs) such as servomechanisms and hill-climbers. These atomic building blocks can be cascaded to form multi-level systems in order to generate sophisticated goal-oriented behaviors with extensible capabilities for adaptation and learning. Purpose, adaptation, and learning cannot be added as afterthoughts, but must be embedded in the core of every module.

The *purpose structure* in an agent incorporates an evaluation function that enables it to tell whether a change is beneficial or not. Setting that evaluation function is an essential aspect of adaptive control. In general, an agent or an adaptive unit within an agent is primarily controlled by a user or another agent setting (or modifying) that function. The units have other control elements as well, e.g., parameters such as step size, thresholds for responses, and gains. Typically, these change much less rapidly and frequently than the explicit purposes.

Within each agent there are sub-agents that undertake the control of lower level agents, all the way down to the control of external actions. At every level each agent or agent structure is undertaking to express and achieve some goal or purpose by controlling its own substructures: the primary control operation is in setting or modifying the purposes of these substructures. These purposes and goals may change with time. They are subject

to the control of higher-level agents and the system users. This approach will ultimately lead to the analysis of the structural components required to enable the system, on its own, to construct new agents that can adapt effectively to address new tasks.

No methodology existed for creating purpose-driven adaptive agent systems. Our project was expressly designed to address this deficiency. Our thesis is that adaptability and learning cannot be added as afterthoughts, but must be embedded in the core of every module. The key idea is that at every level each agent or structure is undertaking to express and achieve some goal or purpose by controlling its own substructures, each with its own lower level purposes. All these purposes and goals may change with time and the whole system is subject to continuing adaptation. The purposes and goals are themselves subject to the control of higher-level agents and the system users themselves.

Key Ideas

The principles underlying our architecture are for the construction and use of purpose-driven software agents are as follows:

- Each agent consists of an adaptive structure of interacting adaptive components and a set of fixed software tools. Each operates in a changing environment that imposes new needs on the agents, and potentially valuable performance benefits.
- Each agent is represented by its *purpose*, the evaluation function by means of which it undertakes its continuing adaptation. That purpose will often in effect be an amalgam of separate purposes or sub-purposes—e.g., to communicate with other agents, to maintain data and check their consistency with other data, to suggest and evaluate plans—all running side by side.
- The agent sub-purposes will be processed by substructures consisting themselves of adapting components, each component being one of a small class of primitive adapting units (EAMs).
- Each component and substructure is controlled by other ones. The essence of this control is the expression or modification of the agent's purpose or evaluation function at the appropriate level.
- Initially, human users will build, run, and modify the agents. Ultimately, however, the agent system itself will generate the continuing adaptive processing.

The terms "adaptation" and "learning" are used in varied, and sometimes imprecise and confusing, ways. Our sense of them is as follows. By adaptation, we mean the ability to change behaviors in response to the environment. By learning, we mean the ability to develop new behaviors in response to the environment. We distinguish four levels of agent adaptation and learning, along the lines illustrated in the following figure.

Level 4: Learning-While-Doing (Online Learning of Adaptive Behaviors) • Automatically develop new behaviors while performing task • Requires ability to rapidly find new behaviors suited to new circumstances
Level 3: Offline Learning of Online Adaptations • Automatically develop new behaviors while not performing task, using experience gained while performing task
Level 2: Purpose-Driven Adaptation (EAM Control Structures) • Agent behaviors are based on changing and unanticipated sensor inputs, via internalized, purpose-driven processes
Level 1: Fixed Pre-Programmed Contingent Responses: Baseline Adaptation • Preset condition-action operation, behaviors are not context-sensitive

Figure 1. Levels of Agent Adaptation and Learning

Level 2 is the starting point of our adaptive agent system development, based on EAM structures, as described above. The EAM architecture supports context-sensitive agent control behaviors in response to changes in the operational environment. It enables agents to adapt their behavior purposefully to address unanticipated situations, e.g., encounters by UAV agents of hostile targets of unknown type or location.

We describe first, the application of EAM-based agents to the complex task of adaptive traffic control. Next, we describe the multi-level EAM agent architecture, and the application of EAM-based agent systems to the UAV military surveillance task. In later sections, we describe the structure and application of OffLine and OnLine learning (Levels 3 and 4), which employ Genetic Programming methods to provide additional enhancements to EAM-based agent adaptation and control capabilities.

Intelligent Traffic Control Application

The purpose of this phase of the work was to explore the behavior of complex traffic flows in a simulated environment using traffic lights controlled by continuously adapting algorithms. The active elements of the simulation were injectors, traffic lights and cars. Cars originated at injectors that were configured to introduce cars at different rates, at fixed or random intervals. Traffic lights were controlled by an adaptive algorithm that changed the fixed interval between traffic light changes. Cars were initially modeled as simple agents with a constant speed, with infinite acceleration and deceleration.

We conducted one set of experiments with EAM-based agents in a traffic simulation environment developed using the BBN OMAR modeling framework. The simulation consisted of a set of intersecting roads, with a traffic light agent at each road intersection. The traffic lights operated with a constant period, during which they switched from green to red, then back to green. The time during that period when the light change takes place was a parameter. Each traffic light agent was controlled by an adaptive algorithm. The agents were elementary control units (EAMs) that incorporated a purpose structure, an evaluation function, and a set of control variables.
The traffic light agents were assigned the purpose of minimizing the average delay of cars on the intersecting roads. This delay time was an environment variable that the agents received as an input. The evaluation function used a hill-climbing approach, comparing the current average wait to that of the previous cycle. An improvement in average wait caused the agent to continue modifying the control variable in the same direction. Conversely, a poorer average wait caused the agent to modify the control variable in the other direction. The control variable was the fraction of the light's period during which the light was green for the east/west traffic. The control variables tuned the evaluation function, and included such things as cycle length (how many light periods before the evaluation function was run.)

Preliminary simulations showed that a single traffic light can converge to the optimal solution of allowing the road with the higher traffic rate to have a longer green time. When traffic rates were changed in the middle of a simulation, the light adapted to the new rates and modified the green times appropriately.

In another set of experiments, a simple adaptive algorithm was employed initially. As each car traversed a traffic light, the delay incurred by the car was recorded by the traffic light. These delays were averaged over a sequence of cars. At the end of each sequence the total accumulated delay was recorded, and the traffic light randomly changed its light change interval by ±5 simulation time units. At the end of 10 of these sequences, the interval corresponding to the lowest aggregate delay was chosen as the base interval for

the next set of sequences. From the traffic light's perspective, this simple algorithm could be regarded as a set of experiments on the delay caused by changing its change interval.

The effects of this algorithm were explored in only a few simple configurations, consisting of up to 3 traffic lights and injectors. This obviously focused more on effects centered around individual lights rather than the more interesting second and third order effects that would arise from larger sets of interacting lights and traffic, but was seen as a necessary first step in understanding the dynamics of the system. In all the simple cases explored, which included single cross-sections and two traffic lights controlling traffic on a straight road, this algorithm invariably decreased over time the aggregate delay experienced by the traffic in the system. Unfortunately, but not unsurprisingly, it almost invariably did this by minimizing the traffic light interval used by each light, typically to the smallest value allowed.

The only exceptions to this behavior occurred when regular periodic traffic was generated, enabling the traffic lights to find a matching cycle which permitted cars to pass without delay. Finding this matching cycle in the solution space was, however, a relatively hard task for the lights, since it required them to find not only a matching interval but also a matching phase offset; and this becomes proportionally harder with a purely random algorithm when there are two traffic lights on the same road. The other factor that mitigated against the light finding the correct interval to cause zero delay to periodic traffic—getting closer to the correct period/phase solution usually increases rather than decreases the delay caused by the light.

We started looking at more complicated situations with multiple roads and lights, and more complicated traffic patterns. Further research was also planned for investigating meta-control agents that controlled the tuning variables of the traffic light agents. The main areas of interest at this point were the development of adaptive algorithms that provided a more directed search through the solution space, and investigation of the behavior of these algorithms in much larger traffic systems, where interactions between traffic lights become much more important. We were also interested in investigating adaptive car behavior, requiring more sophisticated handling of the car agents. However, at this point in the TASK program, around the middle of 2001, we were directed to shift our area of research to work on the CAHDE (Control and Adaptation of MAS operations in Heterogeneous Dynamic Environments) Air Corridor Flow Control scenario, and we began work on developing EAM-based adaptive capabilities in UAV agents.

UAV Surveillance Application

In the UAV surveillance task, EAM-based UAV agents are assigned to search for targets, flying close enough to get accurate sensor readings of target ID, target bearing, and signal strength. They continually evaluate their performance in relation to their purposes and take appropriate actions. UAV agents run under the BBN OMAR agent simulation environment employing BBN OpenMap geographical software. The environment includes clusters of multiple targets, some with unknown locations. It also includes objects that the UAVs were not specifically programmed to handle in advance, such as winds and hostile targets. Higher-level adaptive agents (manager agents) coordinate and control the operation of groups of individual UAV agents. The following figure shows a screen display of a typical scenario in the UAV surveillance environment.

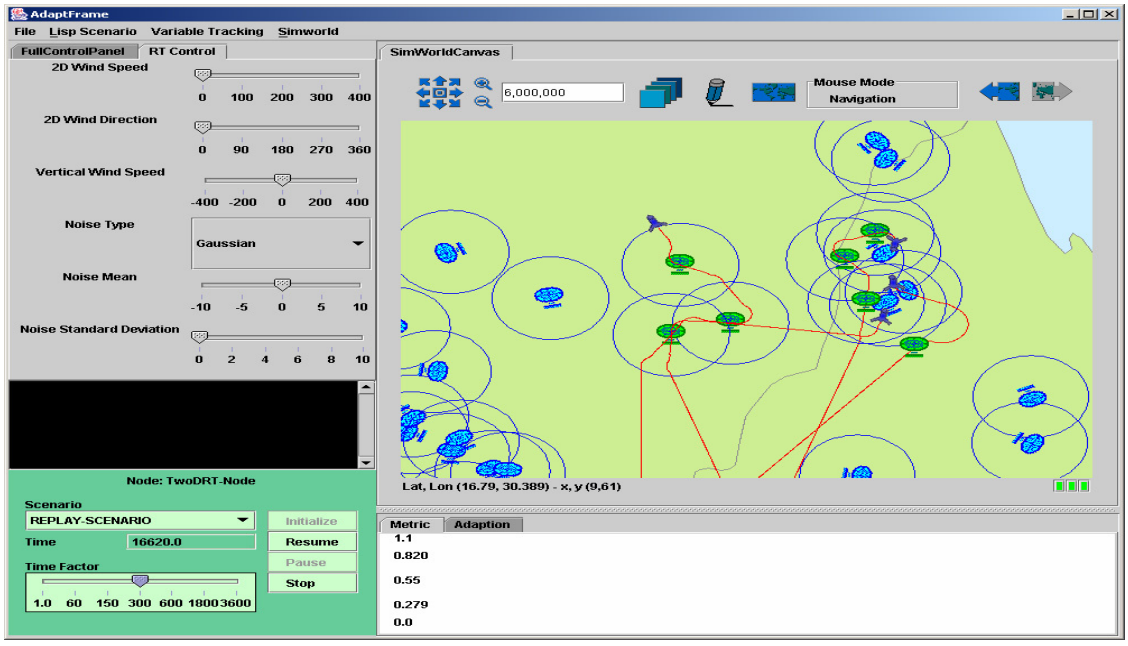

Figure 2. Screen Shot of UAV Surveillance Scenario

In this scenario, a group of 10 UAVs (blue plane icons) is assigned to search an area for targets (blue and green radar icons), and fly close enough to each one to get an accurate sensor reading of the target type and status. The UAVs have noisy sensors that can detect a target and provide the UAV with target signal strength or a target bearing. A hierarchy of elementary adaptive modules (EAMs) controls each UAV. The goal is for the UAVs to discover the status of each target in the shortest possible time. The environment contains objects that the UAVs were not specifically programmed to handle, in advance, such as winds which can push UAVs off course, moving targets, sensor noise, and hostile targets that can down UAVs that come within their firing range.

The operation of a UAV agent is directed by a multi-level control structure of EAMs that interact to provide a variety of adaptive functions. The EAMs are of two major kinds,

servomechanisms and hill-climbers, The archetype of passive EAM systems is the servomechanism, a system such as the household thermostat that uses a setpoint to control the gain of a continuous variable. Servos have been widely studied in classical control theory. *Active* EAMs initiate changes in their behavior, evaluate the effectiveness of those changes, and adapt their behavior accordingly. The archetype of active EAM systems is a hill-climbing process we have termed *Run and Twiddle* (RT), as shown in E. coli, a common bacterium found in the human gut, which moves in such a way as to improve its position with respect to its nutrient. This creature moves, or *runs*, by means of a rotating tail that propels it in roughly a straight line. After traveling for a few dozens of microns, it stops and spins around, or *twiddles*; and then starts off again in another direction, mostly at random. The key to its control is a lesson in simplicity itself: if things get better—detected, say, by an increase in the concentration of the food in its environment—then it keeps on going. The result is that it approaches the source of the nutrient. RT is a flexible and powerful adaptation technique with many engineering applications. It expresses a fundamental practicum of adaptive behavior—keep on doing what works.

Each EAM has six components: a purpose, an action, an action target, an activation, an evaluation function, and a set of control variables. The following list enumerates several of the EAMs in a UAV.

- **Heading Servo.** Activation: periodic. Purpose: minimize heading/signal-bearing difference. Variables: gain, period.
- **Steady Heading RT.** Purpose: minimize delta signal bearing. Variables: increment, period, epsilon. Activation: value triggered: signal-bearing constant (+- epsilon).
- **Gain RT**. Activation: Periodic Purpose: maximize sensor lock count. Action: set gain of heading servo. Variables: increment, period.
- **Speed servo (throttle).** Purpose: match measured-speed set-speed. Error: Function derived with a GA. Variable: gain.
- **Speed RT.** Purpose: maximize speed while avoiding overshoots. Activation: signal. Variables: increment, period. Action: set factor multiplied to set-speed.
- **Signal Merge chooser EAM.** Activation: periodic. Purpose: maximize identified targets. Variables: signals received, targets of other UAVs.
- **Coordination EAM.** Purpose: share useful converged RT data (steady heading, gain, speed) for target-type, Actions: send converged RT data, listen for, ask for reinforcement Activation: periodic.
- **Threat Warning EAM.** Activation: triggered on hostile radar lock. Action: set heading offset.

- **Heading Merge RT.** Purpose: adjust heading to minimize time to find target. Action: combine headings from Heading Servo, Steady Heading RT, and Threat Warning EAMs. Variables: weighting factors of the three EAM components.

These EAMs are structured to form a multi-level control hierarchy, as shown in the following figure.

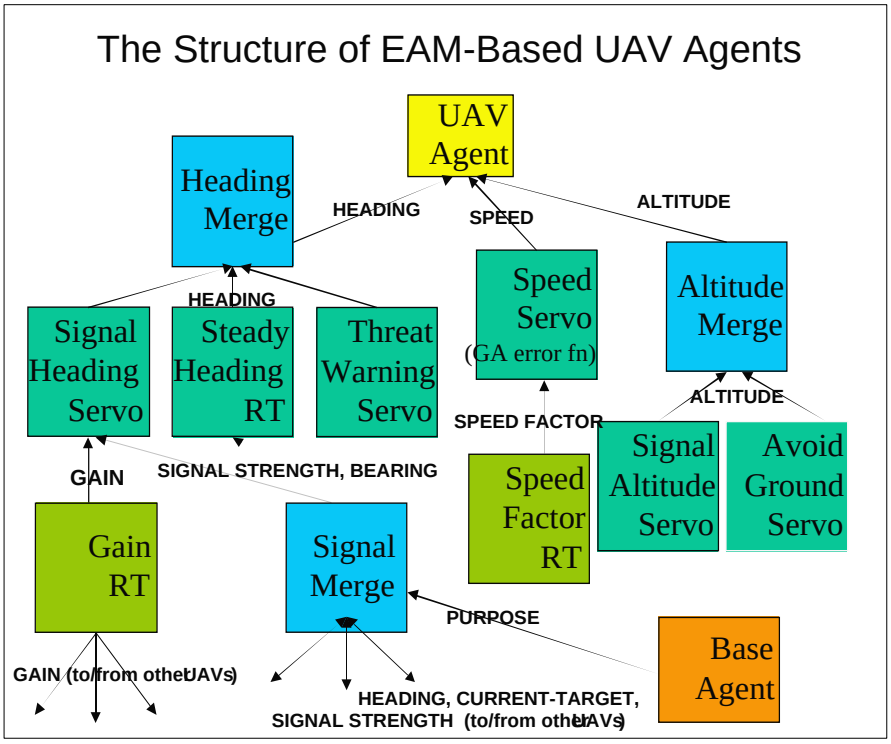

Figure 3. The Multi-Level EAM Control Structure in a UAV Agent System

The joint interactive operation of these EAMs enables UAV agents to behave in versatile adaptive fashion. Higher-level adaptive agents coordinate and control the operation of groups of individual UAV agents. For example, there are four different heading EAMs. The Signal Heading Servo works to minimize the difference between the UAV heading and signal bearing. The Steady Heading RT works to minimize the signal-bearing delta to maintain a steady heading. The Threat Warning Servo seeks to direct the heading away from a detected threat. A higher level EAM, the Heading Merge EAM, continually evaluates the outputs of the other three heading EAMs and computes a weighted average to determine the resulting heading. Another higher-level EAM is the Signal Merge Chooser, which takes as inputs the signals received, headings, and current target locations of nearby UAVs and tries to maximize the number of identified targets. The base agent, a management level EAM, can redirect the Signal Merge Chooser's target priorities.

In a typical scenario, a group of 4 EAM-based UAVs with long-range sensors and optical imaging sensors was tasked to identify 30 targets, 25 with unknown locations. Each UAV was controlled by a hierarchy of EAMs such as that shown above. UAVs had long-range sensors for identifying target type and obtaining readings on signal strength and target

bearing. EAM adaptive heading controls compensated for the effects of winds and moving targets. The graph in the following figure shows the evaluation made by the Heading EAMs and the resulting UAV action in the simulation run.

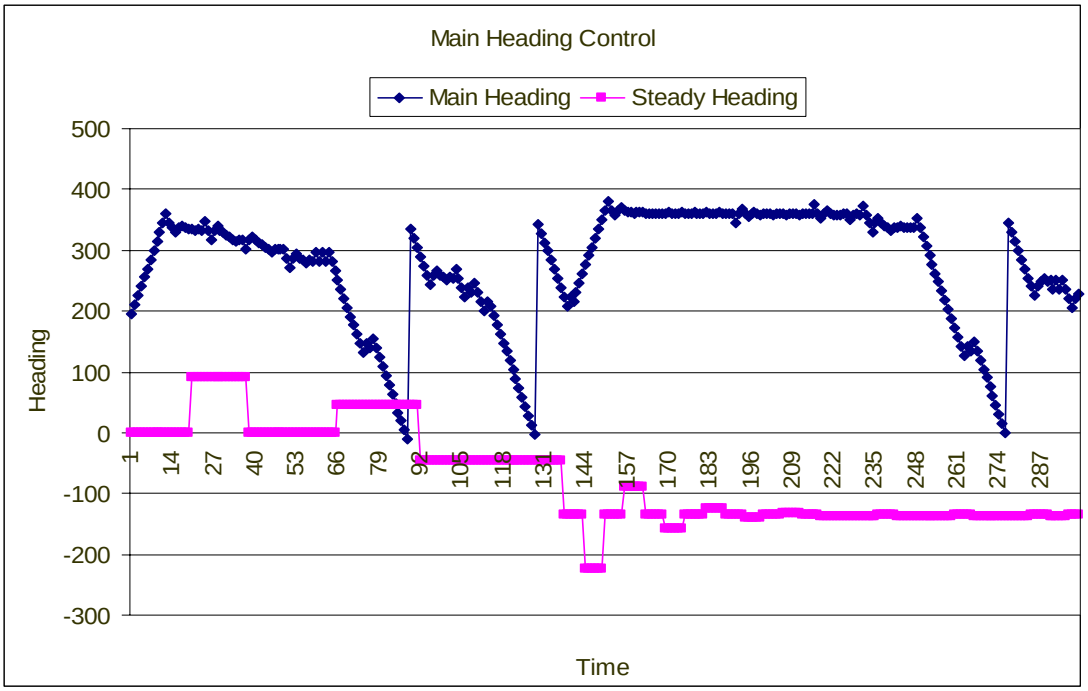

Figure 4. EAM Heading Control: Action and Evaluation

The blue curve shows the UAV heading as directed by the Heading Servo. The slowly decreasing slope during the first minute shows the UAV continually updating its heading, because it is veering off course while trying to locate the first target, The red line shows the effect of the Steady Heading EAM trying to keep the UAV on a straight course by compensating for a strong crosswind that was driving it off course. Before the Steady Heading EAM converges, the Heading Servo EAM is forced to continually adjust the UAV heading (as shown in the decreasing slope of the blue line during the second minute.) When the heading has converged, the UAV is able to fly straight, as shown in the constant slope of the blue line (from around the third minute of the flight), after which it changes its heading to fly toward a second target.

A typical result: without adaptive controls (the baseline Level 1 adaptation described in Figure 1), the UAVs identified 8 targets (3 in unknown locations) in an average ID time of 4243 seconds. With EAM-based adaptive controls, the UAVs identified 11 targets (6 in unknown locations) in an average ID time of 2833 seconds. The UAVs with adaptive controls found twice as many previously unknown targets as the UAVs with non-adaptive capabilities.

The basic EAM adaptive architecture is depicted in the diagram below. The real world

UAV environment is represented in the bottom window. A high-fidelity simulation can substitute for the real world when doing planning rather than controlling actual UAV operations. The UAV sensors can initiate control behaviors directly. When the sensory feedback calls for evaluation of possible responses, appropriate high-level control behaviors are selected from the behavior database to direct the UAV's action. The UAV proceeds throughout the run under the action of this control loop.

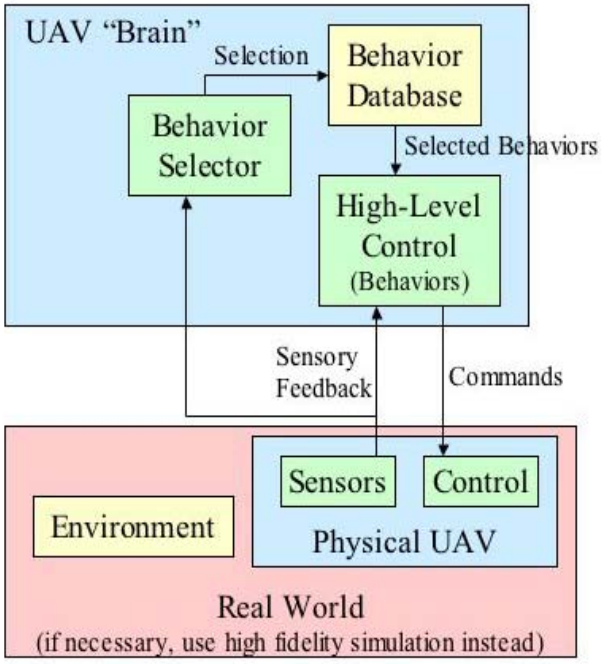

Figure 5. EAM-Based UAV Architecture

EAM controls provide an effective conceptual framework for adaptive control. Building on this framework, basic EAM adaptive capabilities can be further enhanced in a straightforward way. We extended the EAM adaptive architecture by automating a simple idea—learning from experience. Following UAV surveillance runs, we performed off-line learning experiments to develop improved control algorithms for enhancing the UAV's adaptive capabilities.

Agent Learning of Control Algorithms

EAM-based UAV agents have nontrivial capabilities for adapting their behavior in response to anticipated contingencies. For example, they can change the settings of their altitude, speed, heading, or sensor control parameters when the target locations and signal characteristics actually encountered differ from initially anticipated values. However, the agents may encounter situations that were not specifically anticipated, such as signal environments with unusual noise characteristics, or new targets of unknown type. Also, the loss of UAVs to hostile fire can call for rapid generation of new plans for rerouting and re-targeting the remaining UAVs.

To address this problem, we applied genetic programming (GP) to enable a UAV to learn from experience, leading to the generation of more adaptive and responsive control algorithms (corresponding to Adaptation Levels 3 and 4 in Figure 1.) Our object was to provide an automated means of developing improved UAV control algorithms for use by its EAM agents. Moreover, we sought to develop such algorithms "on the fly" in the course of execution of the UAV task. Genetic programming is a technique for learning more effective computer programs, such as control algorithms, for accomplishing a task. It uses a genetic algorithm to search through the space of different computer programs built from a set of primitives, to find the one currently best suited to a given task.

We began by using GP to improve UAV control algorithms "offline", after the execution

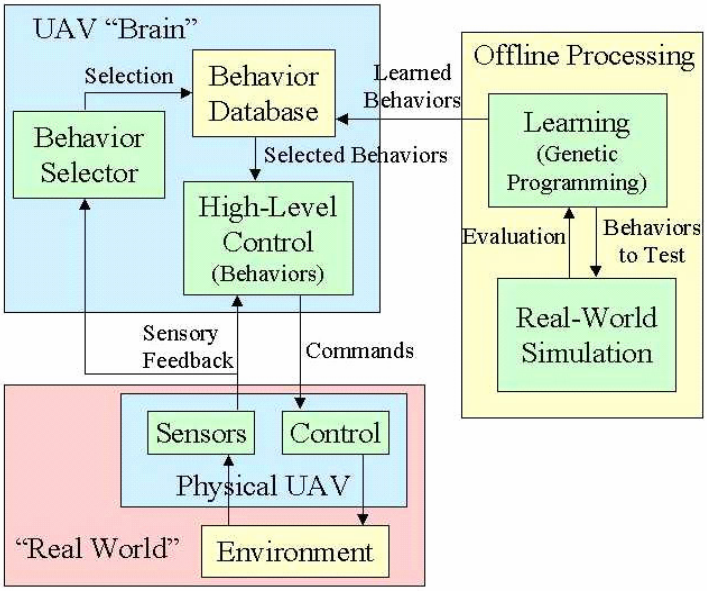

Figure 6. Offline Learning Architecture

of a surveillance run, to better prepare for UAV operations during subsequent runs in response to unanticipated changes in the enemy's capabilities and tactics. Figure 6 shows

our architecture and general approach to *off-line learning*.

This is a schematic description of the Level 3 architecture. It is an extension of the Level 2 adaptive architecture, which is depicted in the two boxes on the left (a repeat of Figure 5 above.) The real world of the UAV and its environment is depicted in the lower box, as before. The off-line processing component shown in the box on the right is the new contribution. The system works as follows. UAV sensors operate through the EAM control algorithms to determine control behaviors. The feedback from these sensors may dictate the need for new control behaviors from the UAV database of previously developed behaviors. To develop new control behaviors, the off-line simulation is run to test existing behaviors under new conditions that may call for new needs. GP learning trials are employed to generate improved control algorithms. These are then embedded in the UAV EAMs for use in subsequent surveillance operations.

New behaviors are learned prior to performing the task and stored in a behavior database. The UAV can adaptively switch between pre-learned behaviors based on environmental conditions, but if an effective behavior is not stored, then it must utilize those that it has. Nevertheless, the UAV can continually improve its control strategy with time by using experiences gained during one period of task performance to develop new behaviors offline before the next such period.

As well as improving its adaptive performance offline in this manner, we would like the UAV to learn to improve its adaptive performance in real-time, while it is performing the task. We refer to such learning in real-time as *online learning* or *learning-while-doing*. However, genetic algorithms (and other learning algorithms) are often poorly suited for online learning, because they generally need to try many poor solutions to find a good one. Such poor behaviors can lead to disastrous effects (such as crashing the UAV!) Furthermore, testing many potential solutions often requires too long a time. Genetic programming requires a search through many possible solutions before finding a more optimal one. Hence, to execute a GP run in a reasonable amount of time requires a reasonably fast way of evaluating possible solutions.

We therefore implemented a fast, highly streamlined version of the full-scale simulator used in the earlier work on the project. The full-scale simulator is a high-level richly instrumented system, serving as a stand-in for the real world. It provides users a testbed for developing, exercising, and refining the current UAV scenario and operational plan. It describes the agents' purpose structures, cognitive functions, and prescribed actions (e.g., UAV routing, timing, and targeting.) It includes the best available descriptors of the hostile entities and the battle environment. It provides user-friendly facilities for supporting plan development, including a graphic user interface and tools for analysis of simulation runs. However, the full-scale simulator has a great deal of overhead associated with display routines, logging of events, and sharing of control between different agents

that the fast simulator does not have. The fast simulator focuses on only that which is needed for the particular UAV agent for which the control is being learned. This allows us to evaluate several different control laws per second.

The highly streamlined rapidly executing simulation serves as a continually adapting "mental model" of the agent task environment. It can be invoked at any time during the execution of the current scenario. It works like this. As agents encounter unanticipated situations (surprises and anomalies) in the course of their operations, these are reported to the mental model. Using the mental model, genetic programming methods evaluate different possible behaviors in response to the new situations and challenges. The solutions thus generated are embedded in the agents. The agents are able to apply their improved behavior during task execution "on the fly" with relatively rapid turnaround. By decoupling the two simulations, we can see the behavior when the mental model does not match the physical world and force the UAV to attempt to modify its mental model in accordance with its observations of the world.

The mental model simulation is several hundred times faster than the high-level simulation. It incorporates genetic programming algorithms for learning to capture the experience of the agents. It applies the algorithms, using the mental model, to evaluate different possible behaviors and generate improved behaviors. These are then incorporated into the high-level model. This enables UAV agent behaviors to steadily improve with experience and to adapt with relatively rapid turnaround time. Moreover, the agents can store learned behaviors for future use in similar situations. Thus, the architecture supports long-term learning as well as short-term adaptation. Figure 7 is a schematic diagram of the online learning architecture (Level 4.).

The UAV continually compares the mental model with what it is experiencing via its sensors, and it updates the model when the model is not consistent with its experience. At its simplest (as in the experiments described below), this is just an adjustment of certain parameters of the simulation, but we envision more flexibility in how to adjust the mental model in the future. Whenever the mental model has changed significantly and there is not a previously learned strategy matched to the current model, the UAV executes a genetic-programming learning algorithm to devise new behaviors suited to the current environment as represented in the current simulation. If the mental model is maintained as an accurate reflection of environment, the learned behaviors should be effective.

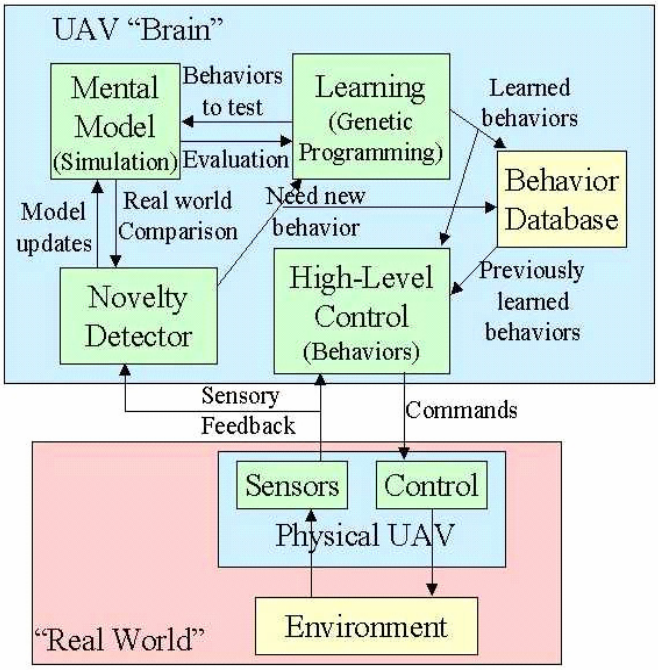

Figure 7. Online Learning (Learning-While-Doing) Architecture

The software used for the genetic programming base was ECJ, a freely available software package developed and distributed by Sean Luke and associates. ECJ supports a variety of different chromosome structures. We applied its standard approach to GP, which utilizes Koza's tree-based chromosome and supporting operators.

The learning algorithm has two main components: the genetic programming infrastructure and the evaluation function. After describing these, we discuss our application of the approach, first to learning bearing control and then, to learning the joint control of bearing and speed.

GP must be customized for a particular problem via the choice of primitives and the evaluation function. We used the following primitives, some that are generic and some that are problem-dependent. The number in parentheses indicates the number of arguments, with zero indicating a terminal, i.e. a leaf in the parse tree of the program:

- CurrentSpeed (0) - the speed of travel of the UAV
- DistanceToTarget (0) - the measured/estimated distance to the closest target
- HeadingError (0) - the (signed) value of the difference between the UAV heading and the measured bearing to the closest target (used only for bearing control and not speed control)
- HeadingDifference (0) - the absolute value of HeadingError
- StandardERC (0) - Koza's ephemeral random constant

- Plus (2) – addition
- Minus (2) – subtraction
- Times (2) – multiplication
- Divide (2) - division (entire expression set to zero if divide by zero attempted)
- IfDistanceWithin (3) - if DistanceToTarget is less than the value returned by the first argument, return the value returned by the second argument, else the value returned by the third argument
- IfHeadingWithin (3) - similar to IfDistanceWithin except using HeadingDifference instead of DistanceToTarget

Two critical parameters controlling the performance of the genetic algorithm are population size and number of generations. For offline learning, where we are willing to wait a relatively long time to obtain a currently best (or nearly best) solution, we used a population size of 20,000 and 20 generations (for a total of 400,000 evaluations). This translates into an overnight run on a standard desktop machine. For online learning, where we need a fast turnaround, we used only 2% of that number of evaluations.

The performance of a control strategy for a single UAV is measured primarily by how many targets it can identify in a particular window of time, which we have taken to be 30,000 simulated seconds. The problem is that a single run provides a very noisy estimate of performance. One source of noise is that a control law can either be particularly well or particularly badly suited to the specifics of the specific scenario used (where the scenario consists mainly of the target placement and the initial position and heading of the UAV). A second source of noise is that, even using the same scenario and control law, the number of targets identified can vary significantly from run to run. This is due to a combination of the noise injected into the simulation (e.g., adding random noise onto the UAV's measurements of bearing) and the "chaotic" way a UAV discovers targets. With only a relatively small number of targets detected during a single run (on the order of 5-15 targets), such noise can be a large problem in terms of robust evaluation.

The solution is to use multiple runs on different scenarios, all generated according to the same statistical specifications (e.g., specifying the statistical distribution of the targets), to perform the evaluation of a control law. By averaging over multiple runs, the evaluation function achieves more statistical significance in its performance measurements. Furthermore, by using a different scenario for each run, we can eliminate a potential source of bias in the performance measurement. There is a tradeoff between using more runs per evaluation, and hence burning more compute time, and using less runs per evaluation, leading to inaccuracy in performance estimates. We have settled on ten runs as a standard number of runs per evaluation.

As a preliminary experiment to validate this approach to learning control laws, we compared the control strategy created by offline learning using genetic programming

with a human-designed control strategy. The human strategy is not “expert” in the sense that it is not applying sophisticated mathematical analysis or the latest control theory techniques; however, it was thoughtfully designed to provide a good measuring stick for “reasonableness”. The procedure was to create ten different scenarios for the mental model simulation as training data. Using these scenarios, we executed the learning algorithm to develop strategies for evaluating fitness for the genetic algorithm. We then created a set of ten other scenarios as test data for the more sophisticated simulation. Using the test data, we compared the performance of the learned strategy with the human-designed strategy.

We developed control algorithms for two different agents: the heading control agent and the speed control agent. The GP chromosome contained two control algorithms, one for calculating a new heading and one for calculating a new speed. To evaluate a particular chromosome, we translated each of these control algorithms into a form executable by the agents and then ran the simulation.

The results generated from the first experiment were as follows:

- The human-designed strategy identified on average 7.2 targets per run on the test set.
- The GP strategy identified on average12.3 targets per run on the test set (and 16.5 targets per run on the training set).

This provided a clear indication that the GP learning algorithm is effective at learning control strategies.

The online learning procedure used the same basic learning algorithm as that used in offline learning. One big difference was the size of the genetic algorithm run in terms of the number of individuals evaluated. For online learning, a fast turnaround of an adequate, albeit suboptimal, solution is better than a long run resulting in an optimal set of control laws. We therefore used a population size of 1,000 and 8 generations, resulting in a total of 8,000 evaluations. Based on trial-and-error with different values of these parameters, this was the smallest number of evaluations that would reliably result in a solution not far from the optimal. (For example, if the optimal set of control laws could identify on average 17 targets, then this procedure might lead to a set of control laws that identified 14 or 15.) With a standard desktop machine, the genetic algorithm required about 7-8 minutes to run, but could run in under a minute with faster hardware.

In further experiments, we employed a scenario in which the radar used to identify the enemy targets has range either less than or greater than that anticipated in the plan, due to environmental circumstances. This required the UAV to learn a new strategy that allows it to fly either closer to, or further from, the targets in order to identify them.

The experimental procedure was as follows. Let R_{id} denote the identification range. Offline, we used the mental model with R_{id} set to value R_0 in order to learn a strategy

tuned to this parameter value. Online, we first execute the "real world" simulation ten times with the parameter R_{id} set to R_1, which is different from R_0, and with the strategy fixed to that learned offline. We averaged the results of the ten runs to evaluate the performance of the fixed strategy. Next, we again executed the sophisticated simulation ten times with $R_{id} = R_1$ but this time with the capability for online learning enabled. When the UAV determined that R_{id} was different from what the strategy was designed to handle, it changed the mental model to the estimated new value of the parameter and invoked the fast version of the learning algorithm to generate a new strategy. The UAV then used the newly learned strategy for the remainder of the run, or until it revised the estimate of the parameter enough that it needed to rerun the learning procedure. (The parameter estimation procedure would continue to operate after executing the learning procedure, and its revised estimates would trigger a second round of online learning.)

We performed this procedure for two different situations: the first with $R_0 = 20$ and $R_1 = 5$, and the second with $R_0 = 5$ and $R_1 = 20$ (i.e., the reverse). The numerical performance results in terms of number of targets identified on average were:

Parameter Values	**Fixed Strategy**	Online Learning
$R_0 = 20$, $R_1 = 5$	1.2	7.4
$R_0 = 5$, $R_1 = 20$	8.4	9.9

The change in the value of the identification range produced some interesting qualitative differences in the learned behavior. For $R_{id} = 20$, the UAV could identify the target in a single pass and then proceed onto the next target. For $R_{id} = 5$, because of the minimum speed constraint, the UAV needed to double back for a second pass to identify a target.

Clearly, the online learning improved performance in these simple test cases. The online learning process required approximately 450 seconds to execute. Since we were assuming in the simulation that the learning process would occur within the 100 simulated seconds of a tick, it is not that far from being real time. With a higher-speed computer, online learning would be fast enough for real-time operation.